

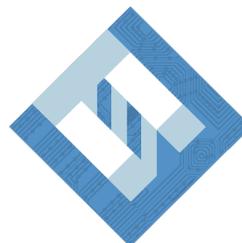
The Case for an Interwoven Parallel Hardware/Software Stack



Kyle Hale, Simone Campanoni, Nikos Hardavellas, Peter Dinda



Northwestern
Compiler
Group



The Interweaving Project
<http://interweaving.org>

The parallel software stack has ossified

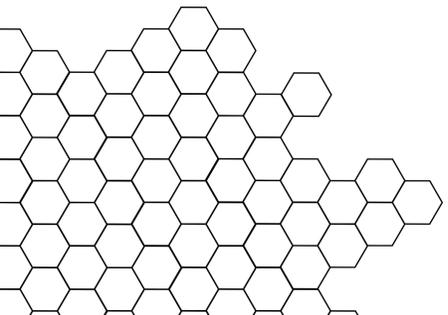
...but

Hardware
Diversity &
Better Design
Tools

Better Compiler
Infrastructure

Virtualization &
Flexibility for
Experimental OS
work

Good position to rethink the parallel HW/SW stack



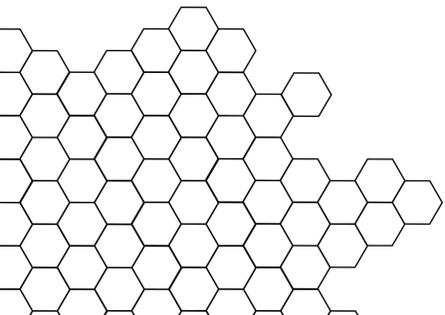
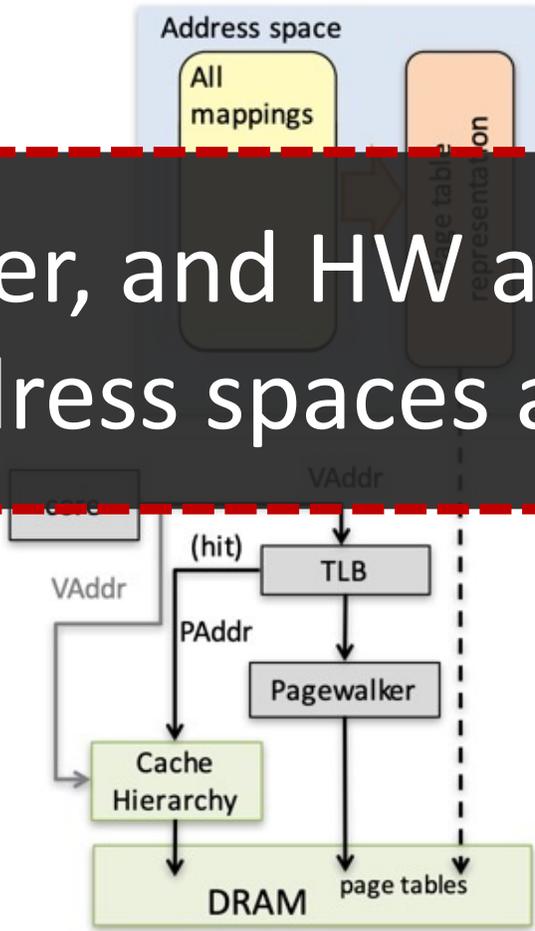
Why Change?

Limitations of the Parallel HW/SW Stack

But OS, compiler, and HW all make assumptions about address spaces and translation!

- + Simplicity for programmer
- + Efficient HW

- Design baked into HW
- Workloads changing
- SASOS increasingly common
- Power consumption



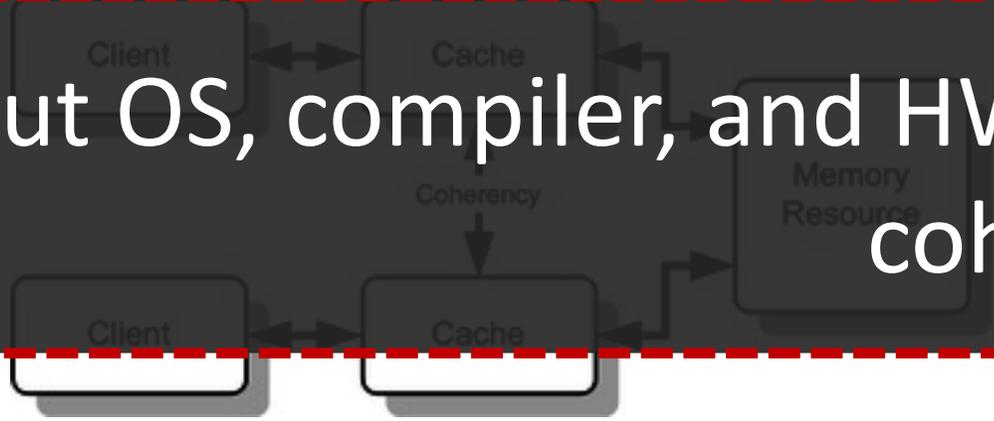
Why Change?

Limitations of the Parallel HW/SW Stack

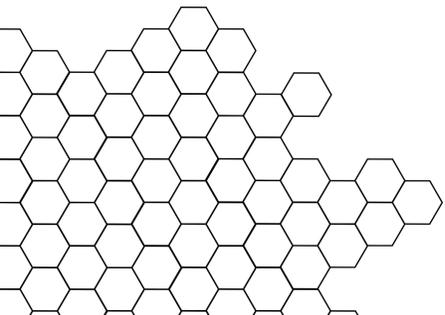
But OS, compiler, and HW all make assumptions about coherence!

+ Enables efficient shared memory
+ Abstraction (hardware "takes care of it")

- Not all workloads need coherence
- False sharing
- No SW control (hidden from OS, language)



HW cache coherence



Many assumptions span the stack...

- Timing

- Synchronization

- Coherence

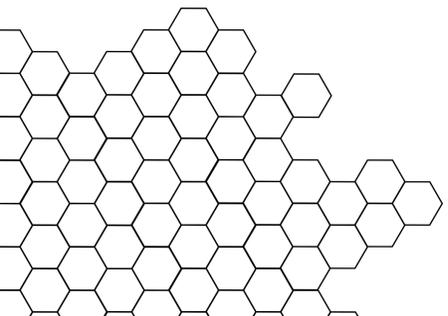
- Address translation

- Execution contexts

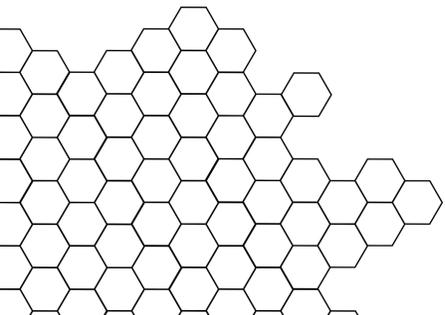
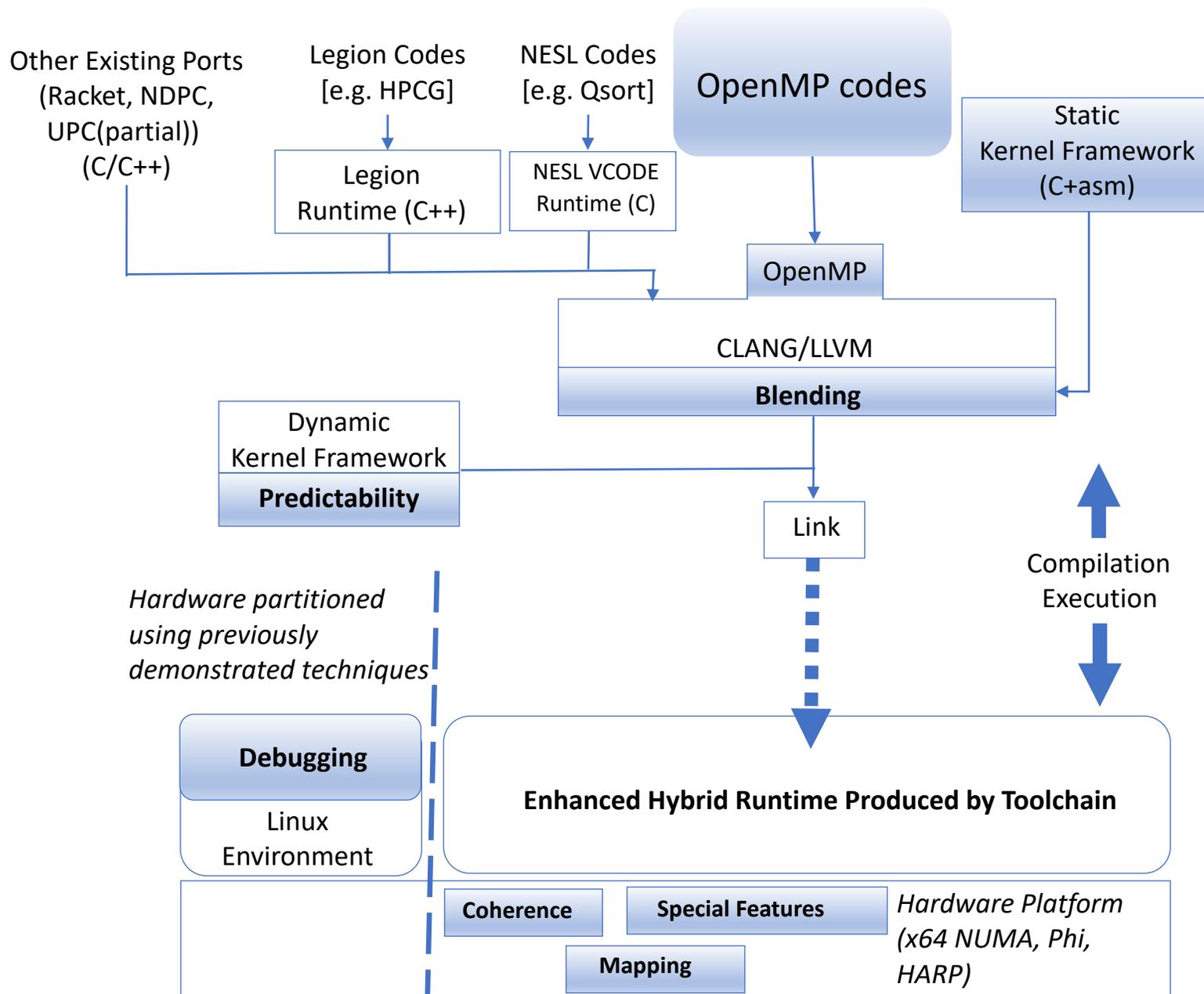
- Scheduling (polling, lack of real-time, etc.)

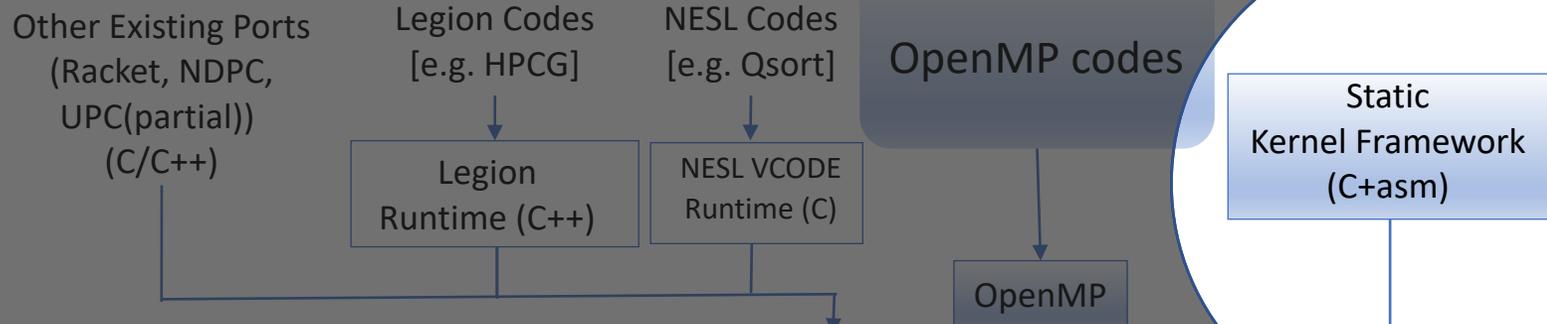
- ...

Interweaving: reconsider these at all layers



System Vision





Nautilus kernel

System Vision

Dynamic Kernel Framework
Predictability

Link

Hardware partitioned using previously demonstrated techniques

<https://github.com/hexsa-lab/nautilus>

Enhanced Hybrid Runtime Produced by Toolchain

Linux Environment

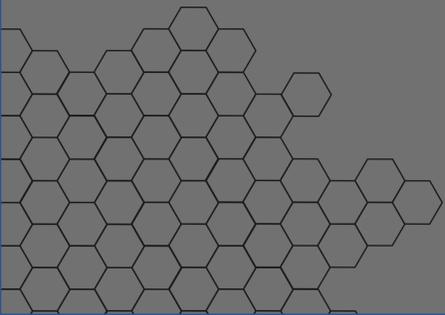
Coherence

Special Features

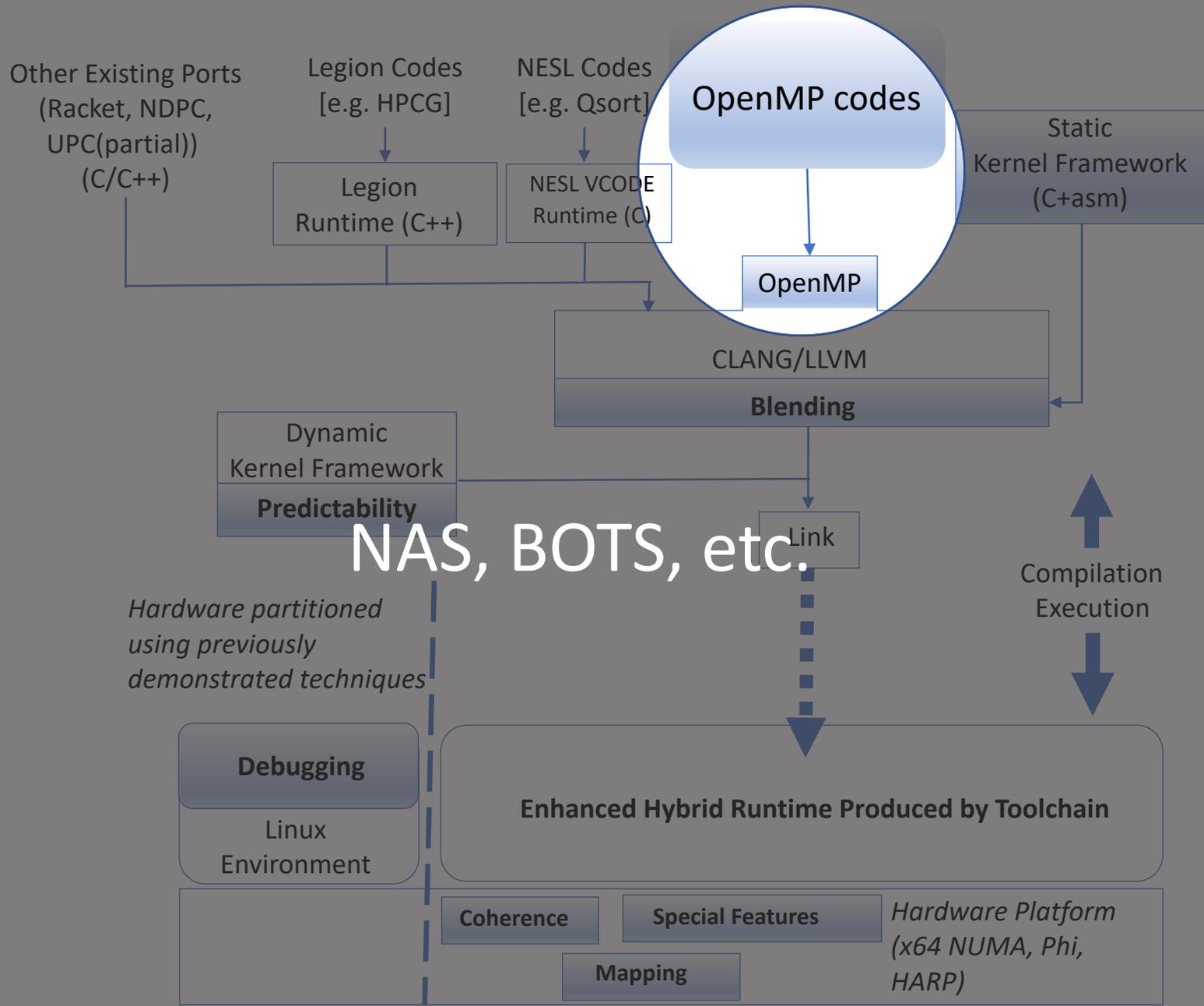
Hardware Platform (x64 NUMA, Phi, HARP)

Mapping

Compilation Execution

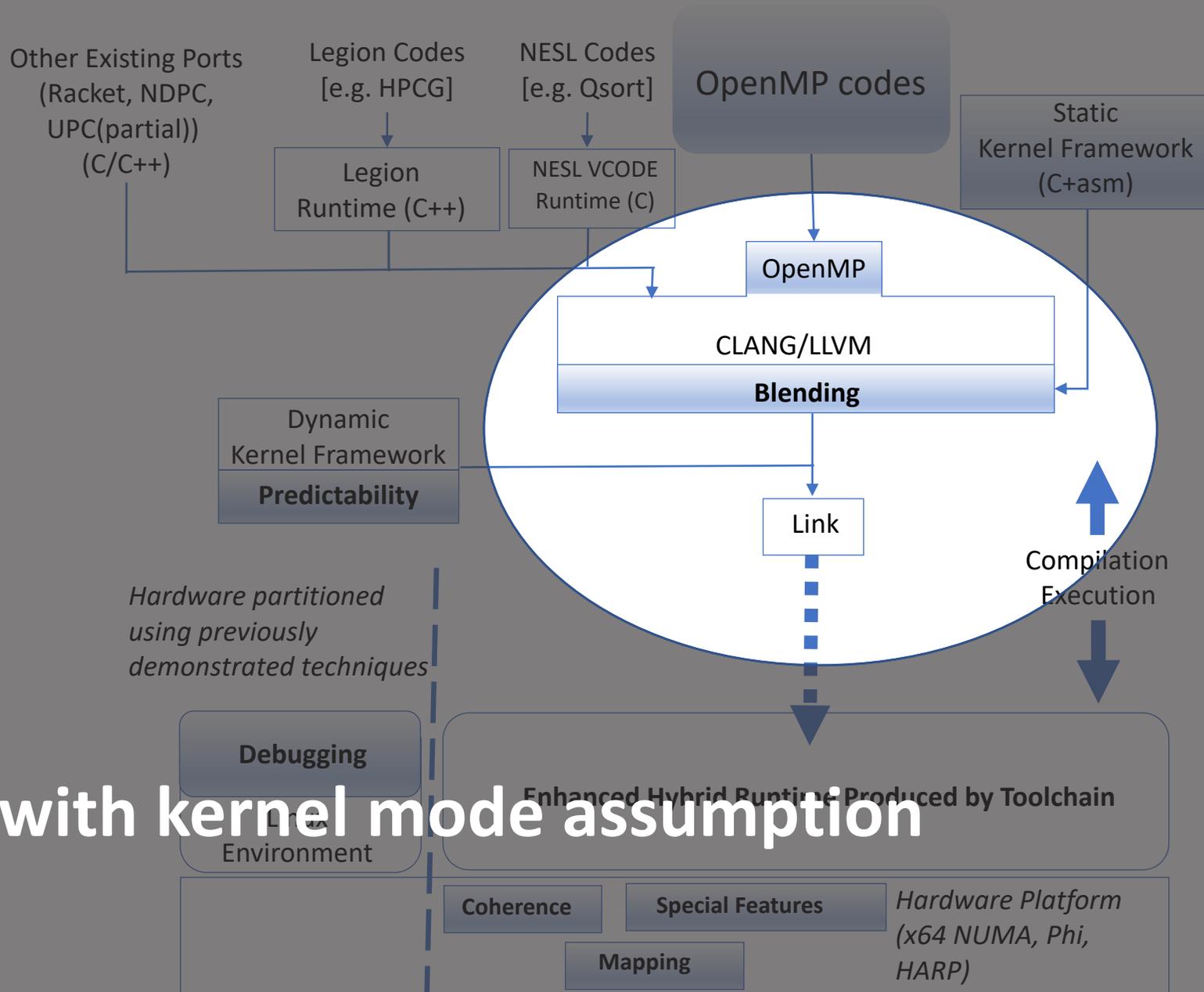


System Vision

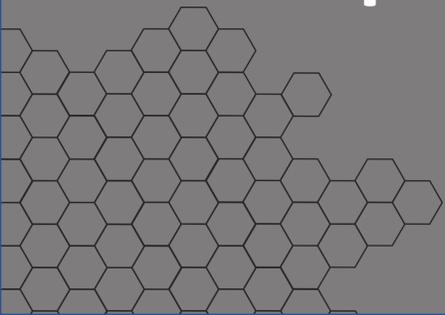


NAS, BOTS, etc.

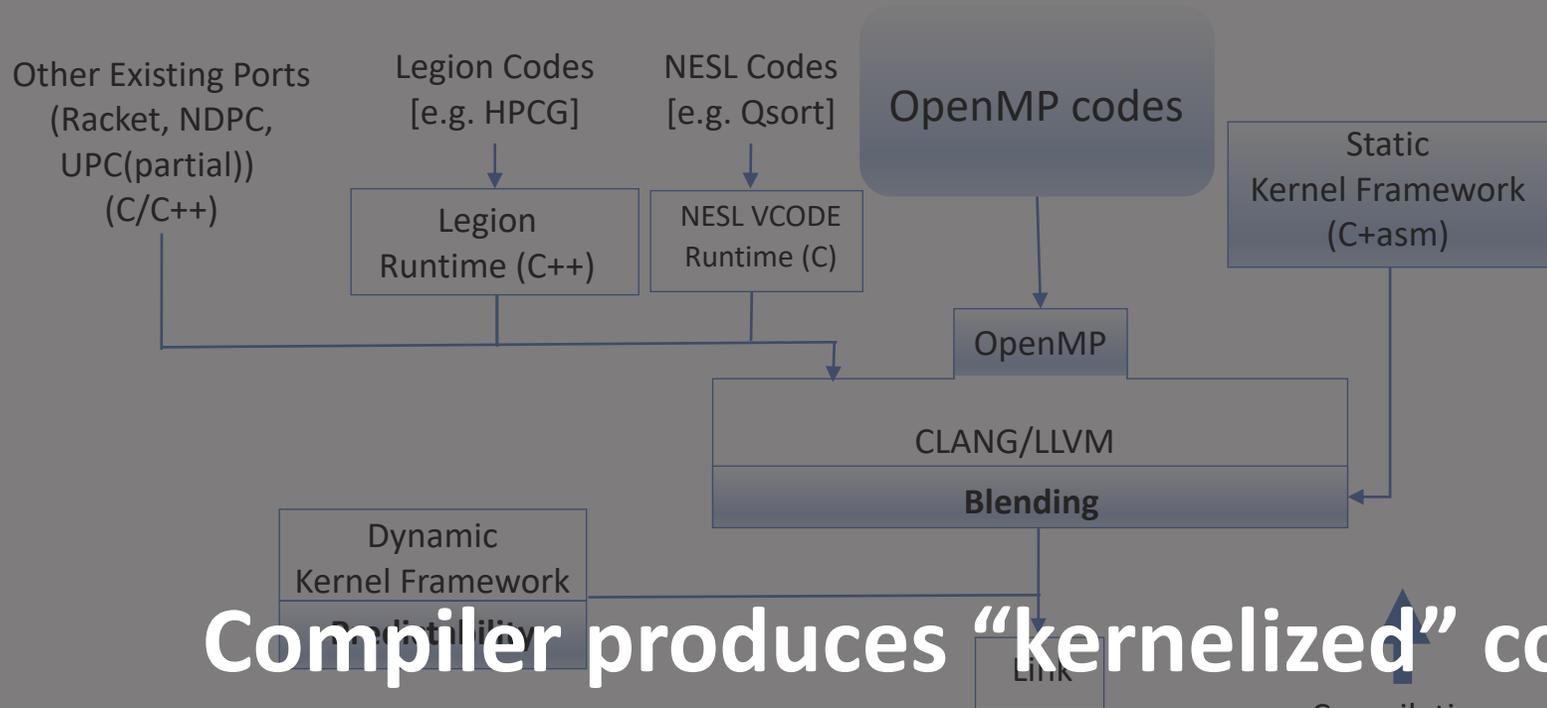
System Vision



Compile with kernel mode assumption

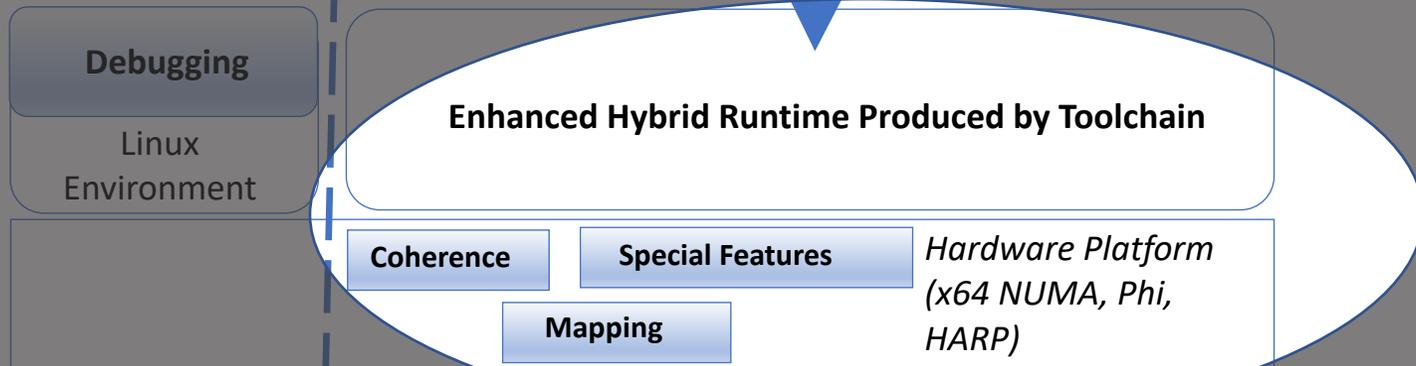


System Vision

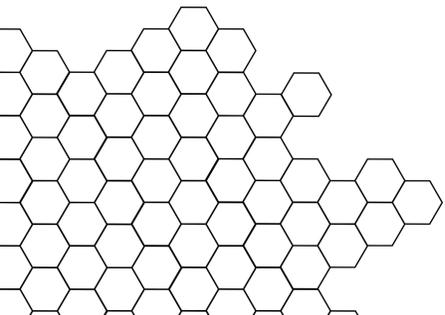
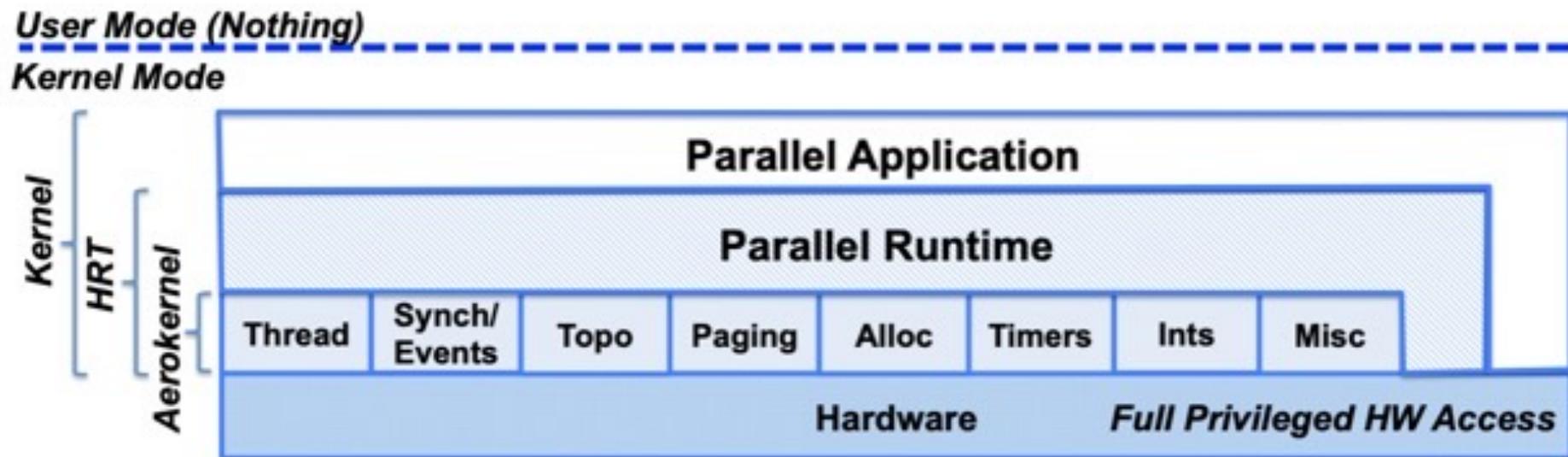


Compiler produces “kernelized” code

Hardware partitioned using previously demonstrated techniques



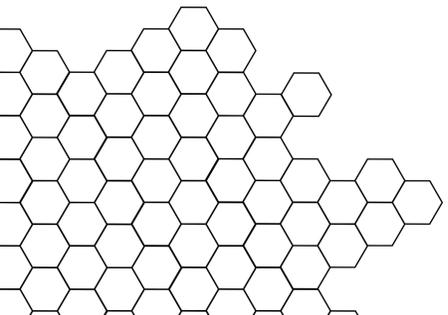
Nautilus Kernel



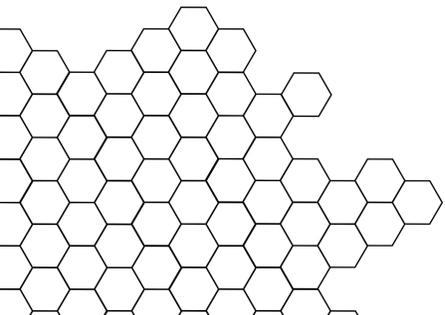
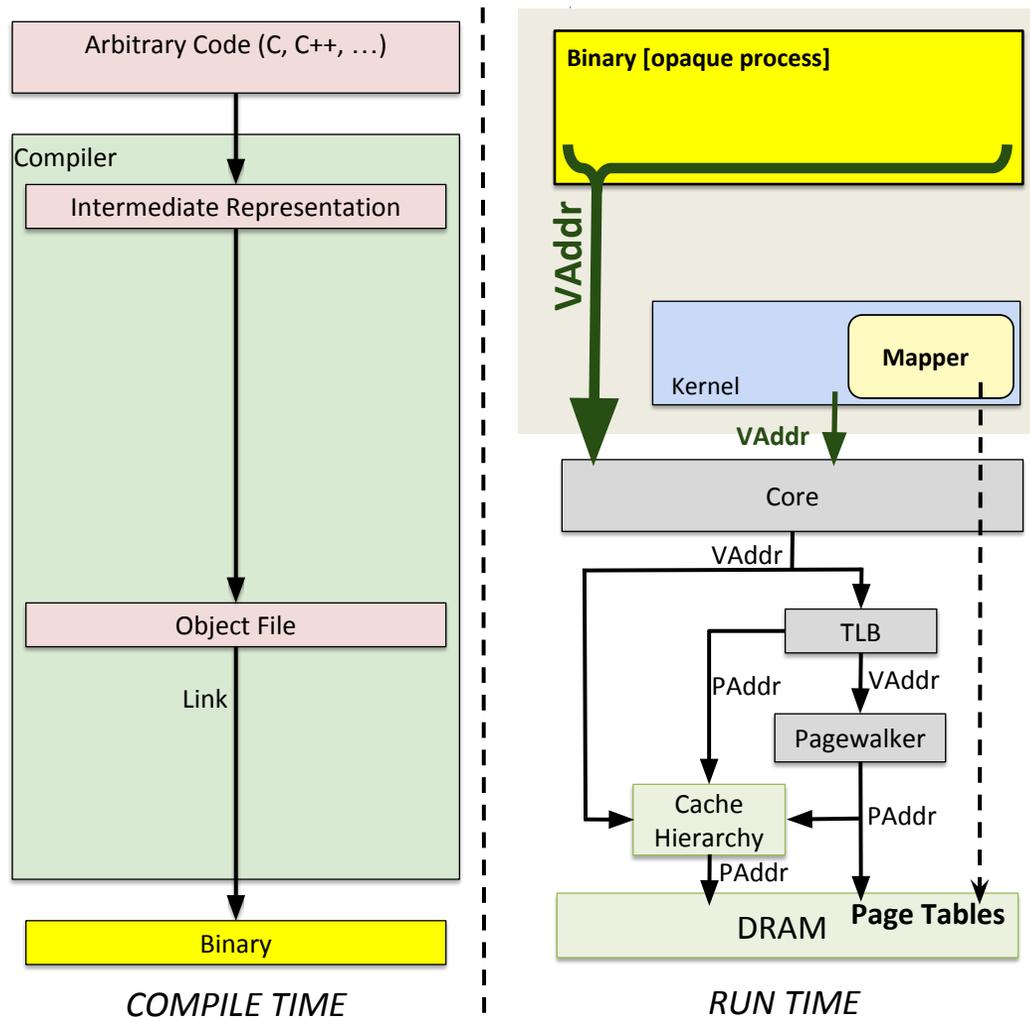
Interweaving Example:

CARAT: Compiler and Runtime-based Address Translation

- Address translation (via paging) is universal, yet showing its age
 - TLB misses are a significant performance inhibitor, including in HPC
 - TLBs/paging limits cache design
 - Fundamentally a **hardware/kernel co-design**
- Can we do better than paging?
- CARAT uses physical addresses instead (no paging, no TLB)
 - Protection and memory migration achieved via **compiler/kernel co-design**
- Proof-of-concept shows feasibility

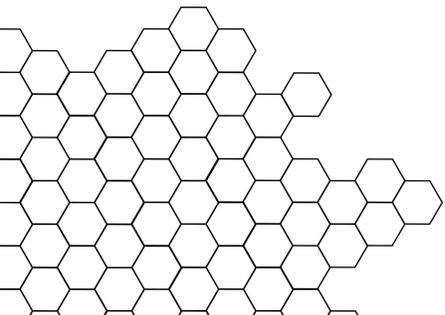
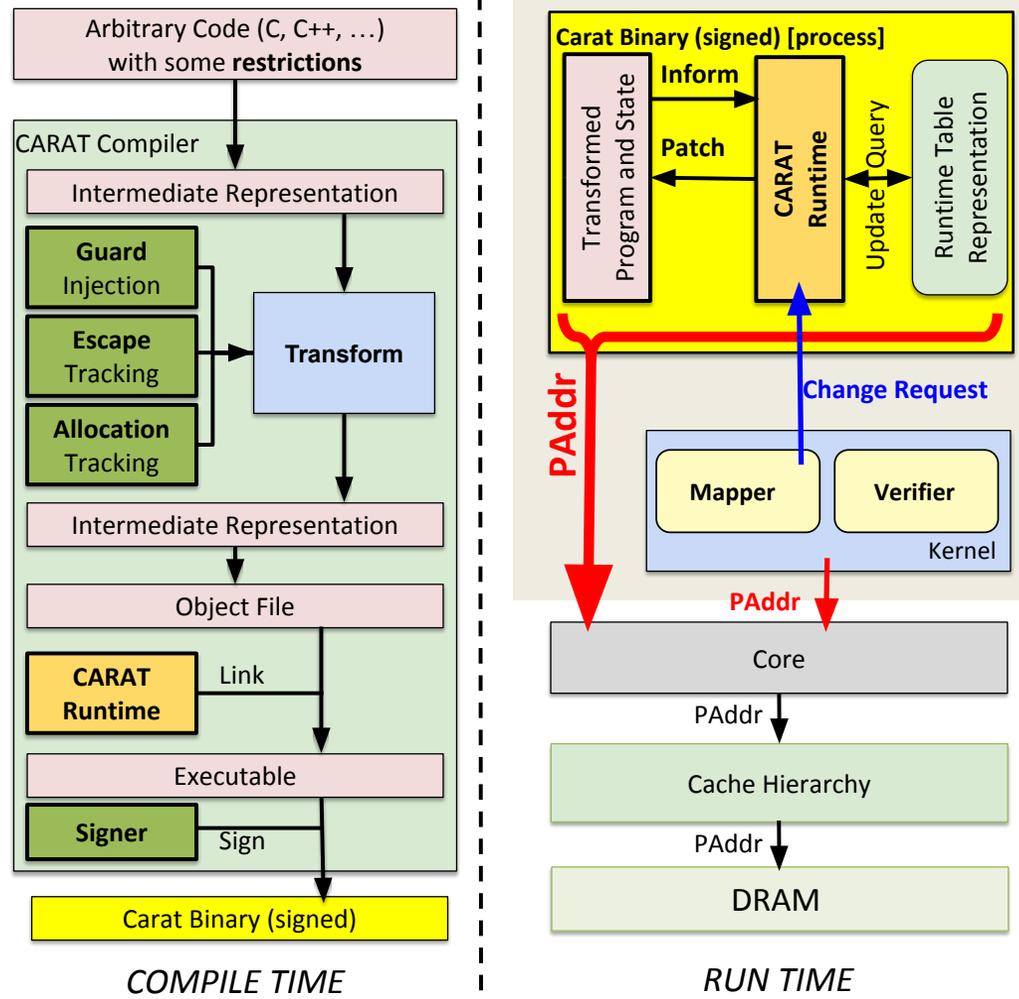


Traditional Address Translation



CARAT: Compiler and Runtime-based Address Translation

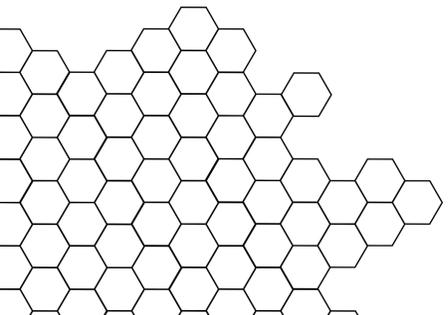
[PLDI '20]



CARAT: Compiler and Runtime-based Address Translation

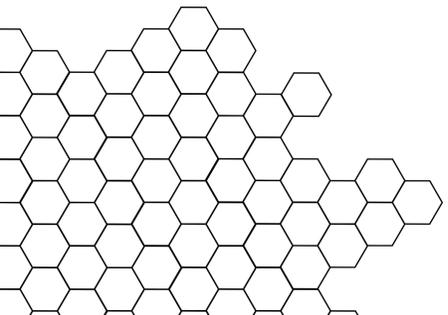
[In Submission Work]

- CARAT integrated into Nautilus Kernel
- Linux-compatible process abstraction...
- ...but **executable runs as a component of the kernel**
- ... yet with protection and memory migration available
- All while using physical addressing

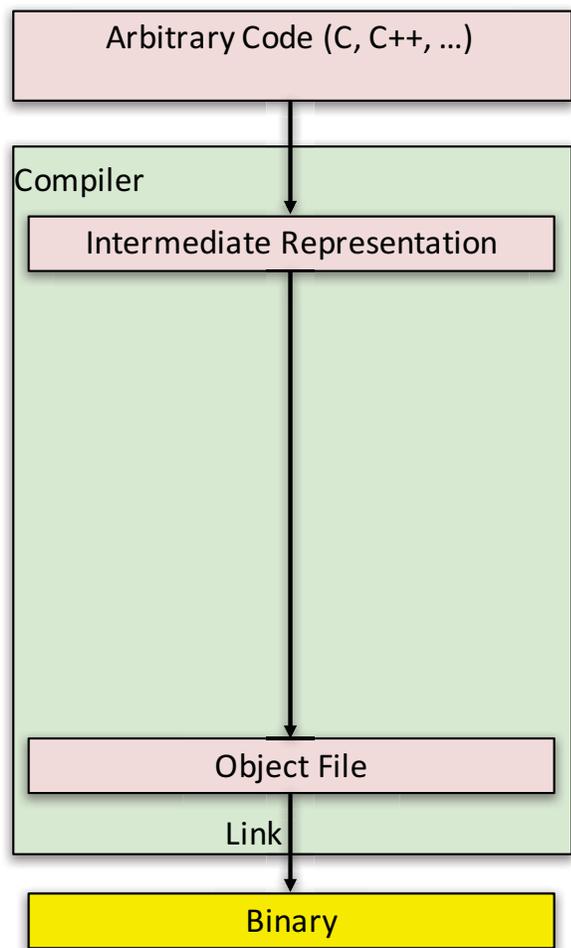


Interweaving Example: Compiler-based Timing (CT)

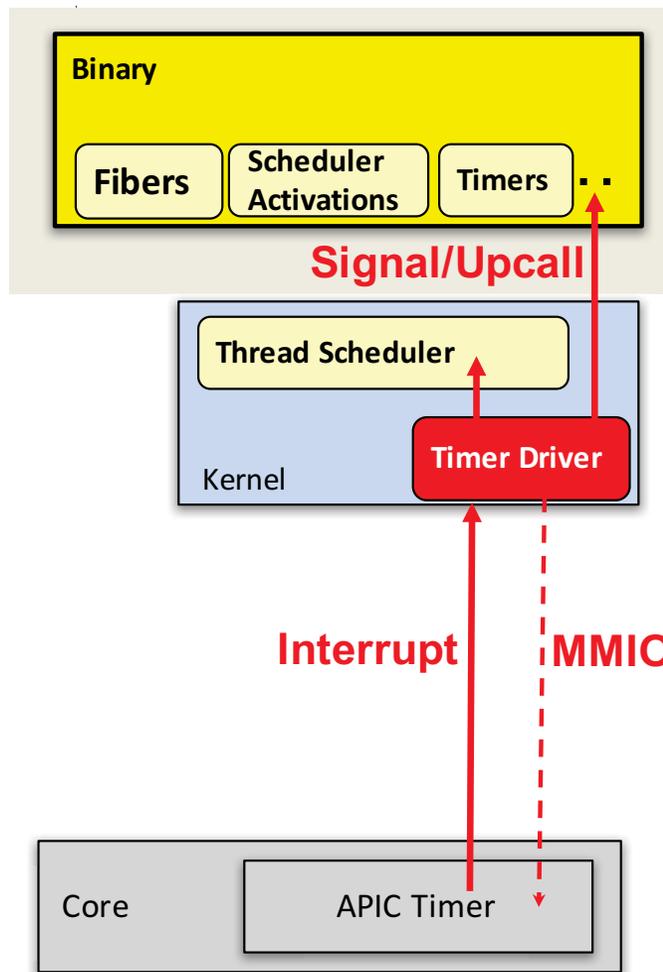
- Timing is traditionally driven by hardware timer interrupts
 - **Interrupt latency is high and not getting lower**
- Limits granularity of many parallel constructs
 - E.g. preemptive threads
- Can we replace hardware timers with callbacks introduced by the compiler *throughout the kernel and application codebase*?
 - Yes. And achieve similar precision. With **6x lower overhead**.
 - Enabling preemptive threads with **4x smaller granularity**



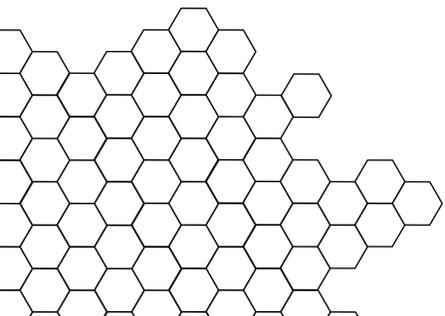
Traditional Timing



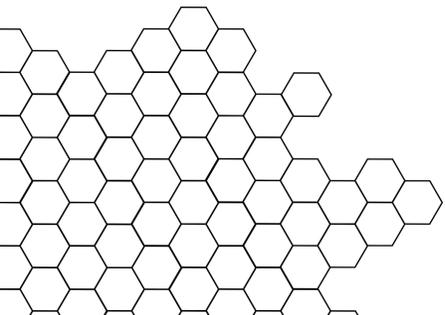
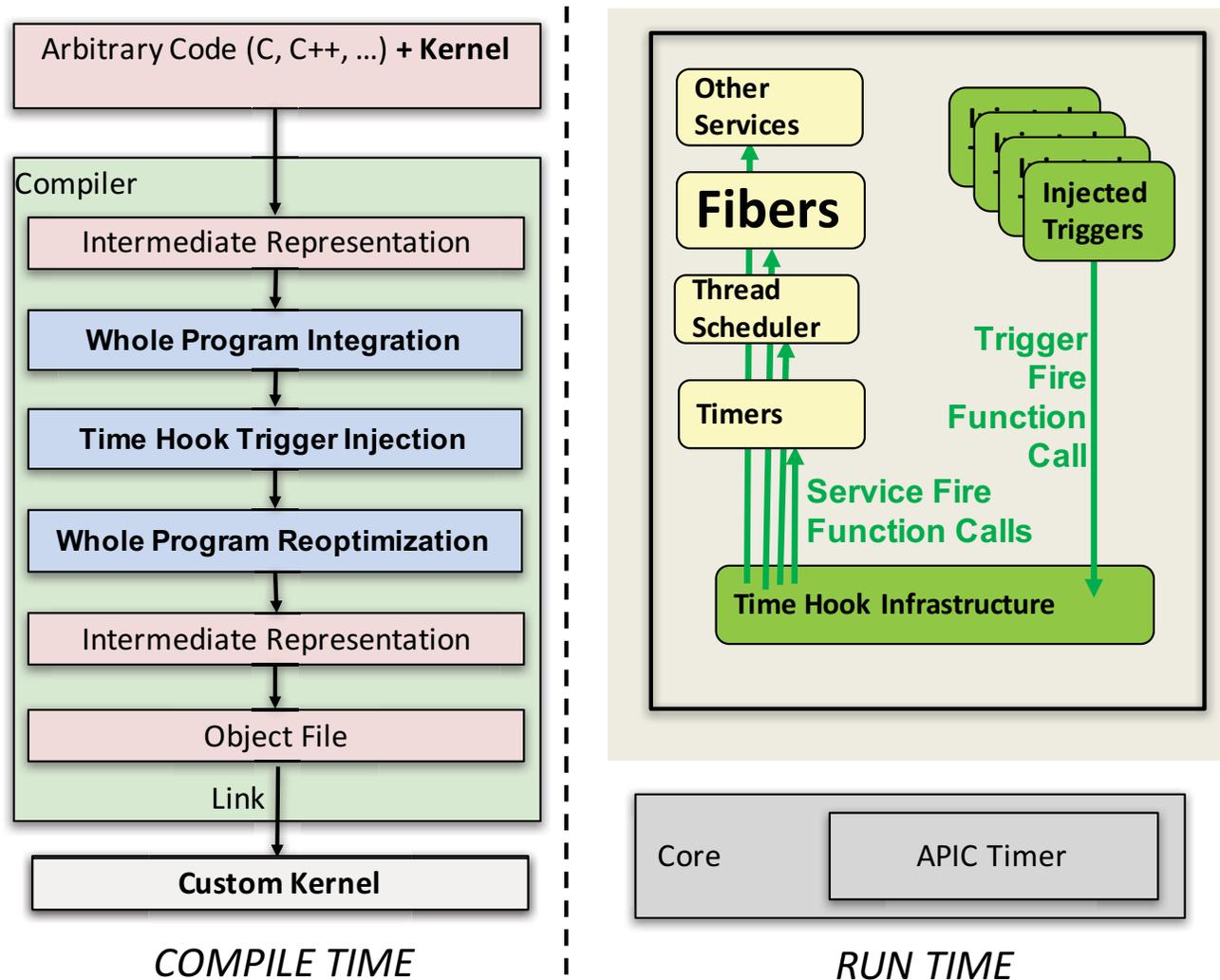
COMPILE TIME



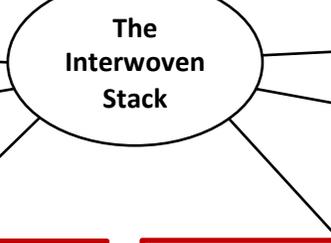
RUN TIME



Compiler-based Timing [SC '20]



Find out more...



A Case for Transforming Parallel Runtimes Into Operating System Kernels

Kyle C. Hale and Peter A. Dinda
Department of Electrical Engineering and Computer Science
Northwestern University

ABSTRACT
Modern parallel runtimes are complex that support a wide range of parallel programming models and languages. This complexity is a result of the need to support a wide range of hardware architectures and to provide a high level of abstraction. This paper presents a case for transforming parallel runtimes into operating system kernels. This approach allows for a more unified view of the hardware and software stack, and it provides a more direct path to hardware acceleration. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

Categories and Subject Descriptors
D.1.7 [Programming Systems]: Operating Systems; D.1.3 [Programming Systems]: Concurrent Programming; D.1.2 [Programming Systems]: Languages

Keywords
Hardware acceleration, RTOS, parallel runtimes, NoFaaS

1. INTRODUCTION
Modern parallel runtimes are complex that support a wide range of parallel programming models and languages. This complexity is a result of the need to support a wide range of hardware architectures and to provide a high level of abstraction. This paper presents a case for transforming parallel runtimes into operating system kernels. This approach allows for a more unified view of the hardware and software stack, and it provides a more direct path to hardware acceleration. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

Prospects for Functional Address Translation

Conor Heffernan, Georgios Trifunakidis, Brian Stuch, Kyle Hale*
*Northwestern University Princeton University Illinois Institute of Technology

ABSTRACT
The need for functional address translation is a result of the need to support a wide range of hardware architectures and to provide a high level of abstraction. This paper presents a case for functional address translation. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

CARAT: A Case for Virtual Memory through Compiler- and Runtime-Based Address Translation

Brian Stuch, Simone Campanoni, Niklas Haderwall, Peter Dinda
Department of Computer Science, Northwestern University Evanston, Illinois, United States

ABSTRACT
Virtual memory is a critical abstraction in modern computing systems. It allows for the execution of programs that require more memory than is available in physical memory. This paper presents a case for virtual memory through compiler- and runtime-based address translation. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

Compiler-Based Timing For Extremely Fine-Grain Preemptive Parallelism

Sourabh Ghosh, Michael Christy, Simone Campanoni, Peter Dinda
Department of Computer Science, Northwestern University Evanston, Illinois, United States

ABSTRACT
Virtual memory is a critical abstraction in modern computing systems. It allows for the execution of programs that require more memory than is available in physical memory. This paper presents a case for virtual memory through compiler- and runtime-based address translation. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

Task Parallel Assembly Language for Unconventional Parallelism

Mike Bailey, Bryan B. Boehm, Kyle Hale, Niklas Haderwall, Simone Campanoni, Peter Dinda
Department of Computer Science, Northwestern University Evanston, Illinois, United States

ABSTRACT
Virtual memory is a critical abstraction in modern computing systems. It allows for the execution of programs that require more memory than is available in physical memory. This paper presents a case for virtual memory through compiler- and runtime-based address translation. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

Paths to OpenMP in the Kernel

Jiepeng Ma, Wenyi Wang, Aaron Nelson, Michael Coates, Brian Stuch, Kyle Hale, Peter Dinda
Northwestern University, United States; Illinois Institute of Technology, United States

ABSTRACT
Virtual memory is a critical abstraction in modern computing systems. It allows for the execution of programs that require more memory than is available in physical memory. This paper presents a case for virtual memory through compiler- and runtime-based address translation. We discuss the challenges of this approach and the benefits of this approach. We also discuss the implications of this approach for the future of parallel computing.

OS + Runtime [HPDC '15]

OS + HW [MASCOTS '19]

OS + Runtime + Compiler [PLDI '20]

Compiler + OS [SC '20]

HLL + OS [PLDI '21]

OS + Runtime + Compiler [SC '21]

Not shown:
Hard Real-time Scheduling for Parallelism (OS+Runtime) [HPDC '18]
Fast Queuing Runtime+Compiler [MASCOTS '21]
Fast Events (HW+OS) [MASCOTS '18]
...and others

see it wednesday

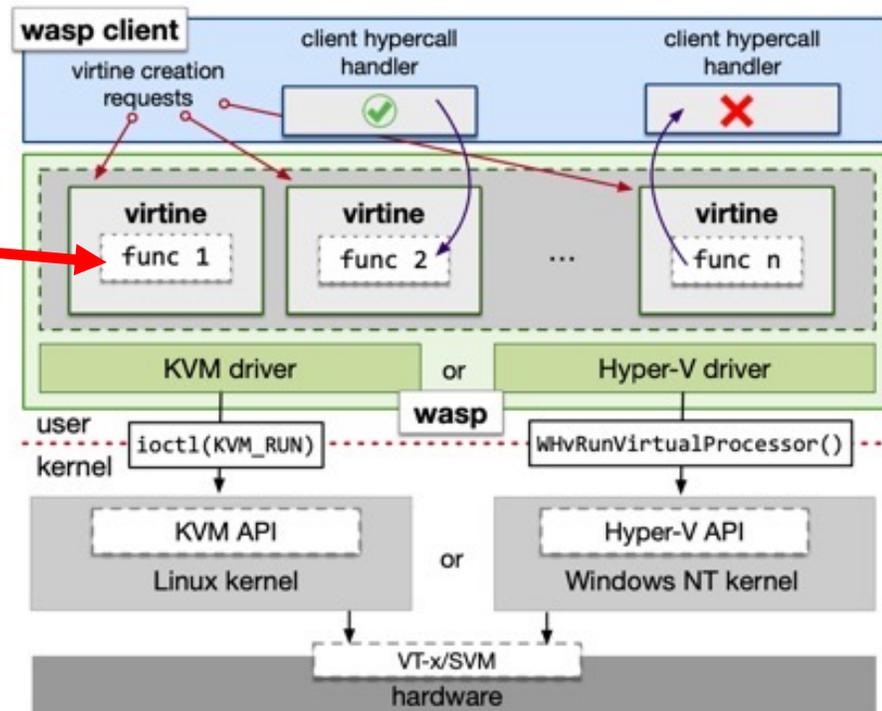
http://interweaving.org

Kyle C. Hale

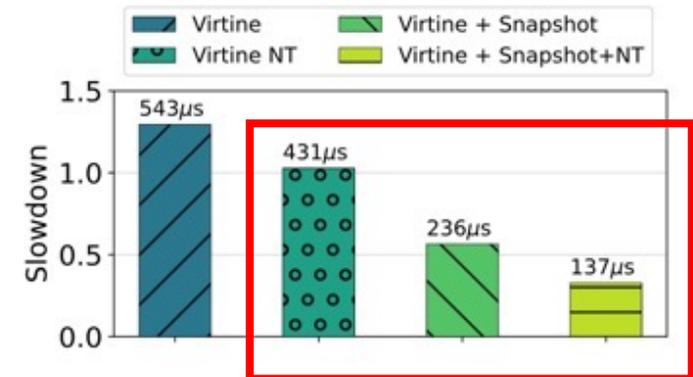
Interweaving Teaser: Function-granularity Virtualization

- More need for single-function execution contexts...even in HPC
- But virtualization platforms not really designed for this

```
virtine int foo() {
    // isolated compute
    return 0;
}
```



Very low overheads for HLL function invocation (FULLY ISOLATED)



Interweaving Teaser: OpenMP

Kernel

Runtime

Compiler

Runtime (Program legwork)

Kernel (see c.)

Runtime

Compiler

Compilation (Kernel compiler does legwork)

Paths to OpenMP in the Kernel

Jiacheng Ma
Northwestern University
United States

Wenyi Wang
Northwestern University
United States

Aaron Nelson
Northwestern University
United States

Michael Cuevas
Northwestern University
United States

Brian Homerding
Northwestern University
Argonne National
Laboratory
United States

Conghao Liu
Illinois Institute of
Technology
United States

Zhen Huang
Northwestern University
United States

Simone Campanoni
Northwestern University
United States

Kyle Hale
Illinois Institute of
Technology
United States

Peter Dinda
Northwestern University
United States

Abstract

OpenMP implementations make increasing demands on the kernel. We take the next step and consider bringing OpenMP into the kernel. Our vision is that the entire OpenMP application, run-time system, and a kernel framework is interwoven to become the kernel, allowing the OpenMP implementation to be customized to the hardware in a custom manner. We consider three approaches to achieving this goal. The first, *runtime in kernel* (RTK), ports the OpenMP runtime to the kernel, allowing any kernel code to use OpenMP pragmas. The second, *process in kernel* (PIK), adds a specialized process abstraction for running user-level OpenMP code within the kernel. The third, *custom compilation for kernel* (CCK), compiles OpenMP into a form that leverages the kernel framework without any intermediaries. We describe the design and implementation of these approaches, and evaluate them using NAS and other benchmarks.

CCS Concepts

- Software and its engineering → Operating systems, compilers; Runtime environments; • Computing methodologies → Parallel computing methodologies; • Hardware → Processors

Keywords

parallelism, OpenMP, operating systems

ACM Reference Format:

Jiacheng Ma, Wenyi Wang, Aaron Nelson, Michael Cuevas, Brian Homerding, Conghao Liu, Zhen Huang, Simone Campanoni, Kyle Hale, and Peter

Dinda. 2021. Paths to OpenMP in the Kernel. In *The International Conference on High Performance Computing, Networking, Storage and Analysis (SC '21)*, November 14–19, 2021, St. Louis, MO, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3458817.3476183>

OpenMP [2, 16, 62] is arguably the most widely-employed approach for the linguistic expression and realization of shared memory parallelism, not least because it extends existing sequential languages like Fortran with parallel features. As a consequence, it can be incrementally adopted. While OpenMP's origins are in compact expression of loop-level data parallelism on SMPs, it has grown to include support for heterogeneous parallelism (including memory and devices), and task parallelism (including fine-grained and recursive tasks).

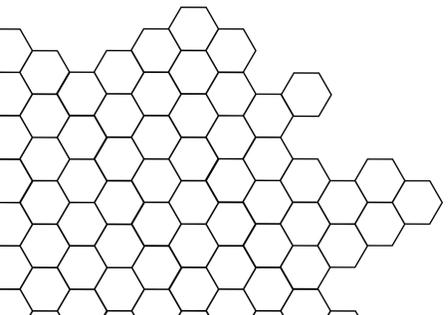
An OpenMP implementation is split between the compiler, which translates OpenMP pragmas (e.g. `#pragma omp ...`) in the context of the sequential host language and lowers them to sequential code, and a run-time system that the lowered code invokes to dynamically manage parallelism. Underneath both lies the kernel, which implements primitives for memory, thread, task, and synchronization management that the run-time system uses, and the compiler uses to generate the most performant way possible.

In a typical implementation, the OpenMP compiler and run-time system target the user-mode process model of a general-purpose kernel. This means that neither the generated code nor the run-time

SC'21
Wednesday, 4PM
(227-228)
Speedups not the point, but, spoiler alert, there are speedups

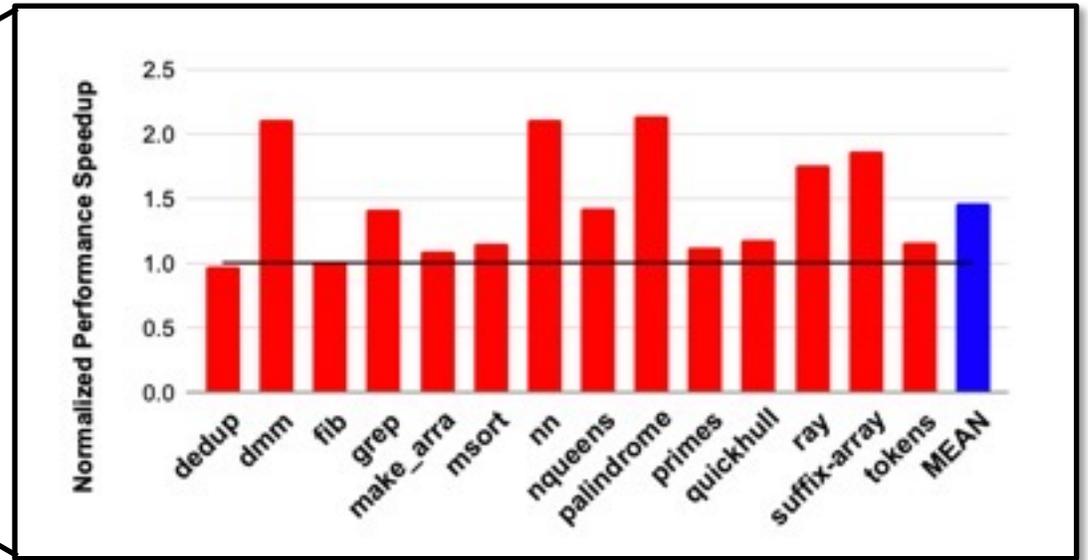
What's Next

- Selective Coherence
- Blended Device Drivers
- Pipeline Interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs



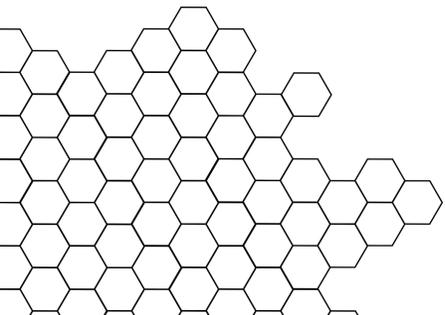
What's Next

- Selective coherence
- Blended Device Drivers
- Pipeline Interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs



Using high-level program information to inform coherence protocol:

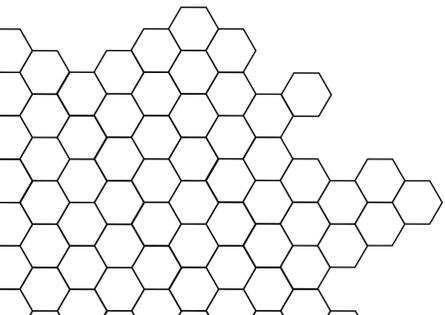
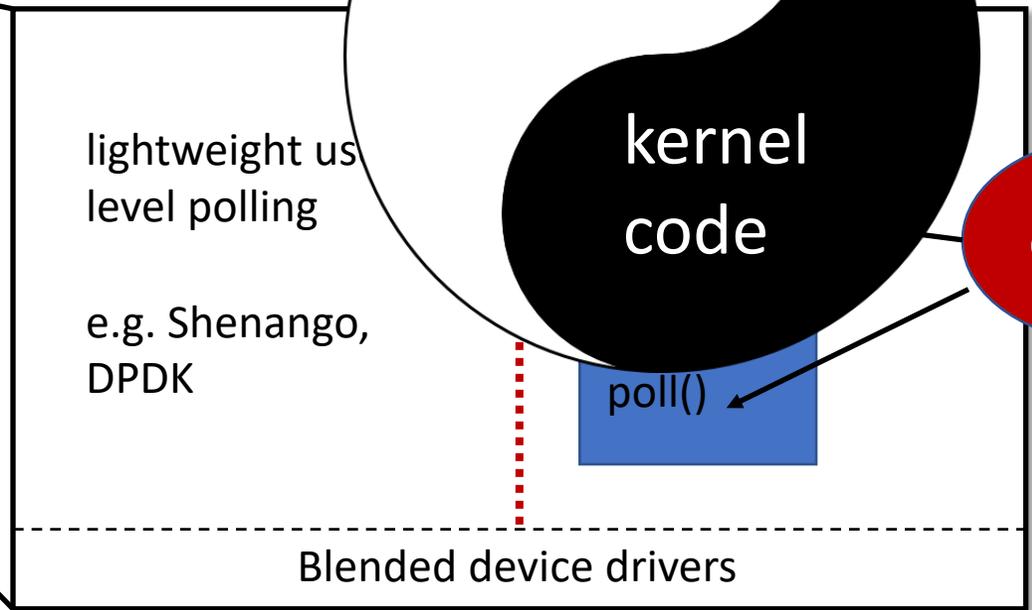
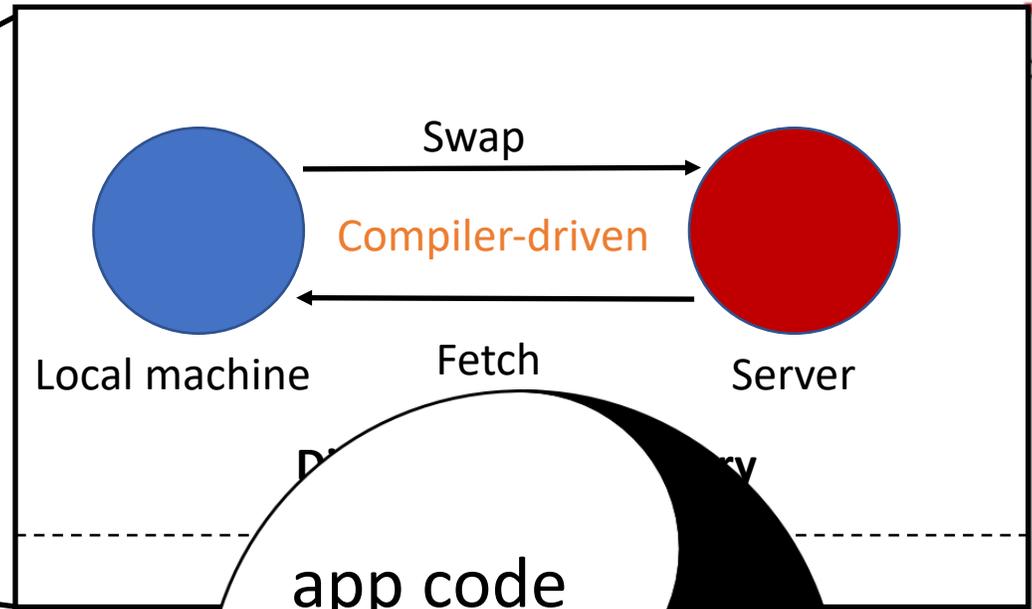
- ~46% speedup
- ~53% interconnect energy reduction



What's Next

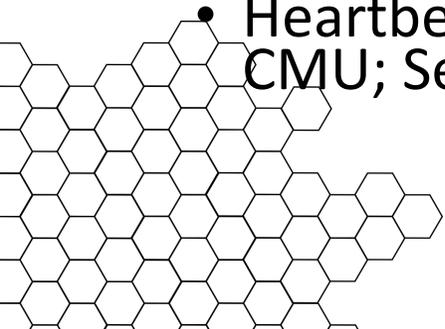
- Selective coherence
- **Blending**
- Pipeline interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs

Interrupt-free Systems



Students and Collaborators

- Many students have or are contributing to these efforts
 - Complete list: <http://interweaving.org>
 - You should hire them!
- Work presented here:
 - Brian Suchy, Mike Wilkins, Souradip Ghosh, Brian Homerding, Jiacheng Ma, Wenyi Ma, Michael Cuevas, Zhen Huang, Conghao Liu, Brian Tauro, Nick Wanninger, Josh Bowden, Enrico Deiana, Vijay Kandihah, Drew Kersnar, Alex Bernat, Gaurav Chaudhary, Siyuan Chai, Kevin McAfee, Kevin Mendoza Tudares
- Heartbeat is in collaboration with Umut Acar, Mike Rainey, and Ryan Newton at CMU; Selective coherence is in collaboration with Umut Acar and Sam Westrick



Check out our OpenMP paper @ SC!
Wednesday, 4PM (227-228)

Thanks to our sponsors:



Thanks!

Thanks to our sponsors:



The Interweaving Project
<http://interweaving.org>



Northwestern
Compiler
Group

